Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

Розробка системи ранжування редакторів за відгуками авторів (Бекенд, ASP.NET)

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
к.ф.-м.н., Сініцина Р.Б.

_____
(підпис)
“____” _____ 2020 р.

Виконала студентка
Рибак Н. Я.

_____
(підпис)
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

Зав. Кафедри мережних технологій,
проф.,д.ф.-м.н.

_____ Г.І. Малашонок
(підпис)

“____” _____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

Рибак Наталі Ярославівні, студентці 4 курсу факультету інформатики
Тема: Розробка системи ранжування редакторів за відгуками авторів (Бекенд, ASP.NET)

Вихідні дані:
Web-API системи ранжування редакторів за відгуками авторів
Зміст ТЧ до курсової роботи:
Вступ
Частина 1: Аналіз предметної області
Частина 2: Теоретичні відомості
Частина 3: Опис реалізації програмного продукту
Висновки
Список використаної літератури
Додатки

Дата видачі „___” _____ 2020 р. Керівник _____
(підпис)
Завдання отримала _____
(підпис)

Зміст

Анотація

У роботі представлено аналіз ринку видавничих послуг та критеріїв оцінювання спеціалістів у різних галузях. Розглянуто різні алгоритми ранжування та актуальність розробки систем для їх реалізації.

Для прикладу було створено Веб-API системи ранжування редакторів за відгуками авторів. Його основна функція – це розрахунок рейтингу редакторів за обраними критеріями.

У першому розділі наведено аналіз предметної області та існуючих аналогів розробки. У другому описано теоретичне підґрунтя до обраного алгоритму. Третій розділ містить розгорнутий опис створення власного застосунку, аналіз технічного завдання, структуру зданих та застосунку.

Вступ

Побудова рейтингу є розповсюдженою функцією для більшості структур, де проводиться оцінка об’єктів, адже необхідно аналізувати дані відгуки. На сьогодні не існує універсального конструктора рейтингових списків для різноманітних систем, а тому формування алгоритму для ранжування редакторів є неочевидною задачею та матиме велику практичну цінність, адже дозволить аналізувати їх роботу залежно від різноманітних критеріїв.

Метою курсової роботи є розробка веб застосунку для побудови рейтингу редакторів, для досягнення якої потрібно виконати наступні завдання:

1) Здійснити аналіз предметної області та аналогічних веб-сервісів;
2) Визначити алгоритм розрахунку та критерії оцінки;
3) Обрати засоби розробки;
4) Створити та заповнити базу даних відповідно до потреб веб-застосунку;
5) Розробити Web-API;
6) Виконати тестування;

Об’єктом дослідження є алгоритми ранжування даних.

Предметом дослідження є вивчення особливостей оцінювання роботи редакторів та технології ASP.NET та MySQL для розробки веб системи.

Використане програмне забезпечення: ASP.NET Core, .NET Core, Entity Framework, MySQL, MySQL Workbench, Visual Studio Code, Postman.

Робота складається з вступу, трьох розділів, висновку та додатків.

В першому розділі роботи досліджено предметну область та сучасний стан розвитку систем ранжування у сфері видавництв та копірайтингу. Розглянуто наявні аналоги розробки, їх основні обмеження та недоліки. Виявлено основні вимоги до функціоналу, яким повинен бути наділена веб система.

Другий розділ містить теоретичне підґрунтя для створення системи, постановку задачі роботи та розгляд алгоритму та методів його реалізації.

У третьому розділі проаналізоване технічне завдання, визначено основні можливості та функції застосунку, а також засоби їх реалізації. Надано опис

розробки Веб API і його окремих складових, таких як обробка Http-запитів, головні функції, дані та структура програми. Також обрані способи та засоби для створення даної роботи відповідно до сучасних тенденцій у веб розробці.

Розділ 1. Аналіз предметної області

1.1 Аналіз сучасного стану питання та обґрунтування теми

На сьогоднішній день рейтинги переслідують нас усюди. Ми оцінюємо товари, статті, фільми, заклади, тощо. В даній роботі розглядатиметься оцінка людей – редакторів, які роблять свою роботу та ранжуються відповідно до своєї праці. Існують критерії яким має відповідати відредагований текст, тож досить складно оцінити якість виконаного завдання однією суб’єктивною оцінкою автора. Тому необхідна модель ранжування яка дозволить оцінити людину на основі критеріїв чи рангів.

Для ранжування враховуються пріоритетність чи вага кожного з критеріїв; тому такий метод оцінювання є досить обґрунтованим та об’єктивним. Дана система повинна дозволяти менеджерам отримувати актуальну інформацію про продуктивність роботи редактора, а також гнучко змінюватись відповідно до необхідних умов. Оскільки розроблених ресурсів з вільним доступом до функціоналу немає, розробка подібної системи є актуальною.

1.2 Огляд існуючих аналогів розробки

Одним з основних етапів розробки майбутнього застосунку є аналіз вже наявних аналогів, оцінки їх позитивних сторін, а також обмежень та недоліків.

Системи ранжування можна зустріти на будь-якому ресурсі, де представлений рейтинг, це може бути рейтинг товарів магазину, фільмів на сайті чи учнів у школі.

Розберемо наступні приклади:

1) Рейтинг товару

Деякі сайти використовують формулу для обрахунку рейтингу товару, яка не дозволяє підніматися в рейтингу товарам, за які голосували 1-2 рази.

Тобто, якщо за товар проголосували 1 раз, поставивши максимальну оцінку - 5, за інший товар проголосували 10 разів - одна оцінка 4 або дев'ять оцінок 5, то

при обліку середньої оцінки на першому місці буде перший товар, тому що загальний рейтинг - 5, а другий буде нижче по рейтингу. Даний підхід не є коректним. Тому використовується наступна формула:

рейтинг = (Сума (голоси) +31.25) / (count (*) + 10), де 31.25 – стартовий рейтинг.

2) Рейтинг сервісу Кінопошук

Даний рейтинг 250 найкращих фільмів є досить відомим і рахується він наступним чином:

$$Rating = \left(\frac{V}{V+M}\right) x\ R + \left(\frac{M}{V+M}\right) x\ C$$

Рис. 1.1 Формула рейтингу Кінопошуку

, де V - кількість голосів за фільм,

M - поріг голосів, необхідний для участі в рейтингу Топ-250 (зараз: 500),

R - середнє арифметичне всіх голосів за фільм,

C - середнє значення рейтингу всіх фільмів (зараз: 7.3837).

Загалом, більшість сайтів з каталогами використовують ту чи іншу формулу для обрахування рейтингу. Проте основним недоліком є те, що ні одна з них не є універсальною і для деяких задач потрібно вигадувати щось нове.

1.3 Постановка задачі

Для виконання основної задачі роботи потрібно виконати такі пункти:

1) провести аналіз подібних сервісів та визначити основні характеристики та вимоги до функціоналу;
2) ознайомитись з принципами розробки веб-API;
3) сформувати технічне завдання;
4) розробити дизайн, що відповідає тематиці завдання – створення моделі ранжування;
5) реалізувати функціонал системи
6) провести тестування розробленого застосунку;

Система має володіти такими функціональними можливостями:

1) Можливість зберігати та відображати дані про редакторів;
2) Відображення списку галузей, підгалузей та під-підгалузей;

3) Задання пріоритетності критеріїв;
4) Пошук по списку;
5) Фільтрація за галуззю.

Розділ 2. Теоретичні відомості

Перед початком розробки застосунку і вирішення поставленої задачі, потрібно визначитись з методами та підходами до її розв’язку. Аналогів з ідентичною функціональністю знайдено не було, тому було проведено дослідження на тему того який відгук має дати автор на роботу редактора та як обраховувати за ними рейтинг.

Перш за все для побудови моделі ранжування потрібно визначити критерії оцінки робіт редактора з редагування тексту. Основними рисами оцінювання було виділено 3 пункти : якість роботи, комунікація між автором та редактором під час виконання завдання та швидкість її виконання.

Детальніше розберемо кожен з них. Якість роботи визначається такими оцінками:

- дотримання граматичних норм
- відповідність тексту галузі та підгалузі
- збереження смислової цілісності
- зв’язність тексту
- структурованість тексту
- інформаційна насиченість

Кожен з них оцінюється автором від 1 до 5, а сама якість обраховується як геометричне середнє даних оцінок. Швидкість – також обраховується комплексно. Вона дорівнюватиме оцінці автора щодо швидкості виконання, яка також матиме значення від 1 до 5, плюс швидкість оцінена системою – відношення кількості сторінок до днів витрачених на це. Дослідивши дане питання, було з’ясовано, що середнє значення швидкості це 15-20 сторінок на день, якщо редактор працював швидше то значення цього параметру буде одиницею, а якщо повільніше – нулем. Додатковим параметром є коефіцієнт галузі, що збільшує загальну оцінку роботи в певній галузі. Тобто, швидкість = оцінка автора * (((кількість сторінок/кількість днів)>15 ? 1 : 0) + коефіцієнт галузі). Комунікація дорівнює лише оцінці автора і має бути від 1 до 5.

Наступним кроком є знаходження зваженої суми трьох основних критеріїв. Це і буде остаточна оцінка по одній роботі редактора. Далі ж потрібно обрахувати рейтинг за всіма роботами та відносно інших редакторів.

Чітких інструкцій зі створення рейтингу немає, проте можна виділити два основних підходи до цього питання.

1) Нижня границя довірчого інтервалу Уілсона для параметра Бернуллі

$$\left(\hat{p} + \frac{z^2_{\alpha/2}}{2n} \pm z_{\alpha/2}\sqrt{\left[\hat{p}(1-\hat{p}) + z^2_{\alpha/2}/4n\right]/n}\right) / \left(1 + z^2_{\alpha/2}/n\right)$$

Рис.2.1 Формула Уілсона

Де $\hat{p}$ = (кількість позитивних оцінок)/(всі оцінки), n – всі оцінки, Z(α/2) = квантиль нормального розподілу.

Даний метод заснований на позитивних та негативних відгуках. Він дає нам нульове значення як для редактора, який не отримує позитивної оцінки автора, так і для редактора, який лише з'явився у системі, але ще не отримав жодної оцінки, що є не цілком коректним, оскільки це означає, що неоцінений редактор це те саме що редактор з негативними відгуками.

2) Імовірний інтервал Байєса

$$S(n_1, \ldots, n_k) = \sum_{k=1}^{K} s_k \frac{n_k + 1}{N + K} - z_{\alpha/2}\sqrt{\left(\left(\sum_{k=1}^{K} s_k^2 \frac{n_k + 1}{N + K}\right) - \left(\sum_{k=1}^{K} s_k \frac{n_k + 1}{N + K}\right)^2\right)/(N + K + 1)}$$

Рис. 2.2 Формула Байєса

Де N – сума всіх оцінок, K – кількість оцінок, Z(α/2) = квантиль нормального розподілу.

Вищенаведена формула враховує не лише позитивні оцінки, на відміну від оцінки Уілсона, а обчислює рейтинги на основі кількості оцінок і тому ця формула краще підходить для мети відранжування редакторів.

Отже, результуюча модель ранжування виглядатиме так: для кожного редактора буде знайдено всі його роботи. По ним обрахована зважена оцінка за якістю, швидкістю та спілкуванням, включаючи такі фактори впливу як коефіцієнт галузі та ваги кожного з трьох пунктів. А рейтинг дорівнюватиме значенню Байєсової оцінки по всім відгукам.

Розділ 3. Опис реалізації програмного продукту

3.1 Аналіз технічного завдання

Головним завданням є розробка системи для ранжування редакторів за відгуками авторів.

Система працюватиме з даними про редакторів, їх роботи та оцінками за виконане редагування, що були виставлені авторами, а також каталогом галузей у яких вони працюють.

Результатом роботи системи є відранжований список редакторів за заданими критеріями, а також результатами пошуку.

Вона призначена для спрощення роботи адміністрації копірайтингових агенств, збору та аналізу інформації про підлеглих.

Функціональними вимогами системи є:

1) Зберігання даних про редакторів та оцінювання їх робіт
2) Розрахунок рейтингу копірайтерів
3) Задання критеріїв ранжування
4) Виконання пошукового запиту за списком
5) Фільтрація даних за галузями роботи
6) Перегляд інформації про редакторів та їх рейтинг

Система повинна надавати API для отримання рейтингу копірайтерів. Оскільки кількість даних може бути великою, система повинна підтримувати посторінкове відображення контенту.

Воно повинно бути реалізоване як Web API через протокол REST, розроблене на платформі ASP.NET, а також використовувати систему управління базами даних MySQL.

Система повинна надавати інтерфейс користувача, а також API для інтеграції з будь-якою іншою системою, якій потрібна функціональність забезпечувана даною програмою. Спроектований API не має залежати від платформи, щоб будь-який клієнт мав можливість використовувати його, не турбуючись про внутрішню реалізацію. Веб-API також має мати змогу розширяти свої функції

незалежно від клієнтських застосунків, при цьому по мірі розвитку наявні функції не повинні заважати роботі існуючих додатків.

3.2 Обґрунтування алгоритму й структури програми

Система працює наступний чином. Оскільки завданням даної роботи є аналіз та отримання результатів ранжування даних, вся інформація яку містить база даних наявна за замовчуванням. Відповідно застосунок може отримувати лише запити на отримання інформації від клієнтської сторони, тобто get-запити.

Дані запиту передаються через параметри URL і містять дані для пагінації – номер сторінки яку потрібно відобразити та кількість елементів, а також закодований об’єкт з потрібними даними про критерії ранжування – ранги обов’язкових критеріїв рейтингу та ідентифікатори вибраних галузей.

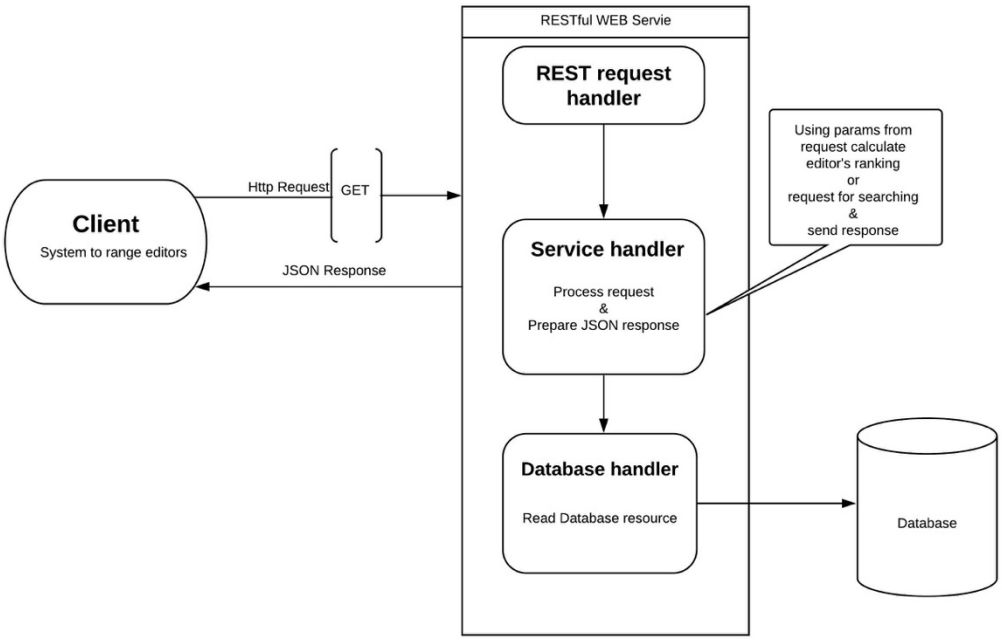


Рис. 3. 1 Структура роботи програми

Далі дані запиту обробляються, розшифровуються за привозяться до відповідних об’єктів системи. Застосунок доступається до наявної бази даних та виконує

необхідний запит, виконуючи необхідні обрахунки рейтингу після отримання інформації про редакторів та їх оцінки.

Результатом є Http-Response у вигляді JSON-файлу, який містить список елементів відображення на вказаній сторінці та розмірність всього масиву результатів ранжування. Ці дані і надаються клієнту для подальшого відображення. Схематичне зображення роботи Веб-API можна представити схемою на рисунку.

3.3 Обґрунтування вибору засобів розробки

Існує великий вибір технологій та мов програмування для реалізації Веб-API. Оскільки однією з умов виконання даної роботи було використання технології ASP.NET, було обрано фреймворк ASP.NET Core, який є її продовженням та працює як на повній платформі .NET Framework, так і з крос-платформенним середовищем .NET Core.

ASP.NET Core включає в себе фреймворк MVC(Model-View-Controller), який об’єднує функціональність MVC, Web API та Web Pages та містяться у програмній моделі ASP.NET Core MVC. З безперечних плюсів даного засобу розробки є функціональність платформи .NET Core, кросплатформенність, розширюваність та модульність.

Відповідно до використовуваного фреймворку, обраною мовою програмування є C#, який є об’єктно-орієнтованим та одним з найбільш потужних та розповсюджених у галузі інформаційних технологій.

Entity Framework є спеціальною технологією на базі фреймворка .NET для роботи з даними. Якщо традиційні засоби ADO.NET дозволяють створювати підключення, команди та інші об'єкти для взаємодії з базами даних, то Entity Framework являє собою більш високий рівень абстракції, який дозволяє абстрагуватися від самої бази даних і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

База даних MySQL - це найпопулярніша в світі база даних з відкритим кодом. Вона є продуктивною, надійною та простою у використанні, а тому найбільш часто використовується для веб-додатків.

3.4 Опис розробки програми

Є декілька підходів до розробки застосунків з використанням Entity Framework, одним з них є так званий «Database-First». Його суть у тому, що спочатку створюються таблиці та поля бази даних, з яких потім буде отримано моделі системи. Для отримання доступу до бази, повинен бути встановлений відповідний провадер, у даному випадку це MySql.

Для спрощення роботи з даними використовувався MySQL Workbench, який забезпечує графічний інтерфейс моделі бази даних та дозволяє легко та швидко модифікувати її, а також вносити необхідну інформацію. Там необхідно задати підключення, наприклад, до локалхосту, ім’я юзера та пароль, ці дані в свою чергу використовуватимуться при зв’язку застосунку з базою.

На рисунку представлена модель бази даних для системи ранжування редакторів.

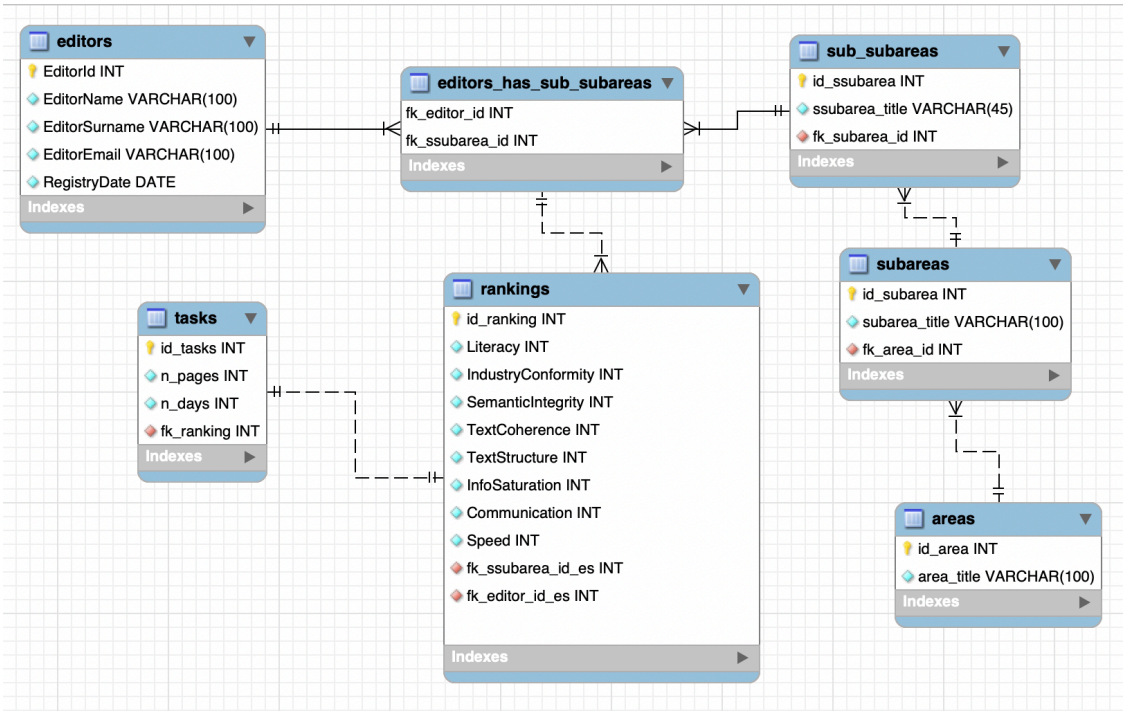


Рис. 3.2 Схема бази даних

За допомогою команди Scaffold-DbContext виконується реконструювання-процес форматування шаблонів класів типів сутностей і класу DbContext (в даній роботі - RankingContext) на основі схеми бази даних та отримаємо класи що відповідають таблицям. RankingContext містить також метод OnModelCreating який описує створення моделей, перелік списків усіх класів у типі даних DbSet, а метод OnConfiguring реалізує підключення до провайдера бази даних.

Для роботи з даними та обробки запитів створюються контроллери, які визначають шляхи та параметри запитів до розроблюваного API.
Основним контроллером застосунку є RankingController.

3.5 Створення об'єктів і розробка головної програми

Основними об’єктами програми є редактори, ранкінги – оцінки по виконаним роботам, власне роботи та галузі. Для всіх них в результаті реконструювання було створено класи-моделі, та також спільний клас-контекст RankingContext.

```
using System;
using System.Collections.Generic;

namespace RankingApi.ModelsDB
{
    4 references
    public partial class Editors
    {
        0 references
        public Editors()
        {
            EditorsHasSubSubareas = new HashSet<EditorsHasSubSubareas>();
        }

        11 references
        public int EditorId { get; set; }
        4 references
        public string EditorName { get; set; }
        4 references
        public string EditorSurname { get; set; }
        4 references
        public string EditorEmail { get; set; }
        1 reference
        public DateTime RegistryDate { get; set; }

        5 references
        public virtual ICollection<EditorsHasSubSubareas> EditorsHasSubSubareas {
    }
}
```

Рис. 3.3 Модель даних редактора

Далі, за допомогою методу OnConfiguring() класу RankingContext та провайдера Pomelo.EntityFrameworkCore.MySql реалізується підключення до бази даних яку було попередньо створено за допомогою MySQL Workbench.

```
0 references
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseMySql("server=localhost;port=3306;database=rankingdb_schema;user=root;password=****",
         x => x.ServerVersion("8.0.19-mysql"));
    }
}
```

Рис. 3.4 Підключення до бази даних

Після цього створюємо клас-контроллер RankingController, який буде виконувати запити до нашого Web API. На рисунку 3.5 показано як вказується шлях за яким доступний сервіс – api/ranking, а також в конструкторі створюється екземпляр контексту, що забезпечує взаємодію з базою та моделями.

```
namespace RankingApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    0 references
    public class RankingController : ControllerBase
    {
        15 references
        private readonly RankingContext _context;

        0 references
        public RankingController(RankingContext context)
        {
            _context = context;
        }
```

Рис. 3.5 Підключення до бази даних

Основний алгоритм ранжування реалізований у методі GetEditorsRating. Разом з get-запитом приходить також стрічка з параметрами, яку необхідно декодувати на привести до об’єкту критерії.

Як було зазначено вище, у цій роботі використовується імовірний інтервал Байєса для вирахування рейтингу кожного редактора по його оцінкам. Але перед

тим потрібно вирахувати саму загальну оцінку по всім критеріям, вказаним у ранкінгу. Для цього ми беремо середнє геометричне значень оцінок з тих характеристик оцінених автором:

- дотримання граматичних норм
- відповідність тексту галузі та підгалузі
- збереження смислової цілісності
- зв'язність тексту
- структурованість тексту
- інформаційна насиченість

Отримане значення умовно позначаємо якістю виконання роботи.

Наступним кроком вираховуємо оцінку швидкості, використовуючи значення, поставлене автором та значення яке рахуємо – це відношення кількості сторінок роботи до днів які пішли на її виконання та множимо на коефіцієнт галузі.

Далі отримані заначення а також оцінку комунікації використовуємо для знаходження зваженої суми – це і буде наша оцінка.

Такі перетворення необхідно зробити для кожного ранкінгу редактора, щоб потім з масиву даних значень обрахувати рейтингову оцінку.

Повертаючись до реалізації, необхідно спочатку отримати весь масив з даними по редакторам, тобто всі підгалузі в яких вони працюють, всі ранкінги та роботи. Для цього було виконано запит, який повертає список об'єктів класу AreaRankingTaskDto:

```
var allinfo = ( from r in _context.Rankings
                join t in _context.Tasks on r.IdRanking equals t.FkRanking
                join se in _context.EditorsHasSubSubareas on r.FkEditorIdEs equals se.FkEditorId
                join e in _context.Editors on se.FkEditorId equals  e.EditorId
                join ss in _context.SubSubareas on se.FkSsubareaId equals ss.IdSsubarea
                join s in _context.Subareas on ss.FkSubareaId equals s.IdSubarea
```

```
                where r.FkSsubareaIdEs == se.FkSsubareaId &&
criteriaObject.ssubareas_checked.Contains(ss.IdSsubarea)
                select new AreaRankingTaskDto{
                    IdRanking = r.IdRanking,
                    Literacy = r.Literacy,
                    IndustryConformity = r.IndustryConformity,
                    SemanticIntegrity  = r.SemanticIntegrity,
                    TextCoherence = r.TextCoherence,
                    TextStructure = r.TextStructure,
                    InfoSaturation = r.InfoSaturation,
                    Communication = r.Communication,
                    Speed = r.Speed,
                    IdTasks = t.IdTasks,
                    NPages = t.NPages,
                    NDays = t.NDays,
                    EditorId = e.EditorId,
                    IdArea = s.FkAreaId,
                    IdSsubarea = ss.IdSsubarea,
                }
        ).ToList();
```

Далі групуємо цей список по EditorId та в циклі прораховуємо значення Rate – рейтингової оцінки. Метод

```
private double FinalRate(AreaRankingTaskDto[] rankings, List<AreaK> area_coefs,
double[] coefs){
        List<double> ranks = new List<double>();
        foreach (AreaRankingTaskDto r in rankings){
           double[]marks = {r.IndustryConformity, r.InfoSaturation, r.Literacy,
r.SemanticIntegrity, r.TextCoherence, r.TextStructure};
           double quality = GeomAverage(marks);
           double communication = r.Communication;
```

```
            double area_coef = area_coefs.Find(x => x.id == r.IdArea).area_k;
            double speed = Speed(r.NPages, r.NDays, r.Speed, area_coef);
            double weightedSum = WeightedSum(quality, communication, speed, coefs);
            ranks.Add(weightedSum);
        }
        foreach(double rank in ranks){
            Console.WriteLine(rank);
        }
        return Rate(ranks.ToArray());
    }
```

Метод що повертає значення рейтингової оцінки за масивом переданих значень:

```
private double Rate(double[] rates){
        double N = rates.Sum();
        int K = rates.Length;
        double z = 1.959963;
        if (N == 0) return 0;
        double first_part = 0.0;
        double second_part = 0.0;
        for(int k=0; k < K; k++){
            first_part += (k+1)*(rates[k]+1)/(N+K);
            second_part += (k+1)*(k+1)*(rates[k]+1)/(N+K);
        }
        double score = first_part - z * Math.Sqrt((second_part - first_part*first_part)/(N+K+1));
        return score;
    }
```

3.6 Тестування програми і результати її виконання

Одним з умовних етапів розробки веб-ресурсу є його тестування, виявлення помилок, похибок у роботі.

Проведемо тест розробленого Веб API за допомогою застосунку Postman.

1) Перший запит лише на ранжування за критеріями:

http://localhost:5000/api/ranking?page=1&page-size=2&criteria=%7B%22quality_k%22%3A0.4%2C%22speed_k%22%3A0.3%2C%22communication_k%22%3A0.1%2C%22areas_k%22%3A%5B%7B%22id%22%3A1%2C%22area_k%22%3A0.3%7D%2C%7B%22id%22%3A2%2C%22area_k%22%3A0.25%7D%2C%7B%22id%22%3A3%2C%22area_k%22%3A0.2%7D%2C%7B%22id%22%3A4%2C%22area_k%22%3A0.15%7D%2C%7B%22id%22%3A5%2C%22area_k%22%3A0.1%7D%5D%2C%22ssubareas_checked%22%3A%5B0%2C4%2C5%2C7%2C8%5D%7D

Отримаємо результат


```
{
   "size": 6,
   "editors": [
      {
         "editorId": 3,
         "editorName": "Katya",
         "editorSurname": "Tkach",
         "editorEmail": "tkach@gmail.com",
         "rate": 1,
         "editorSubSubareas": [
            "Audiology",
            "Emergency Medicine"
         ]
      },
      {
```

```
            "editorId": 4,
            "editorName": "Ilya",
            "editorSurname": "Evdokimov",
            "editorEmail": "evdokimov@gmail.com",
            "rate": 1,
            "editorSubSubareas": [
                "Cardiology"
            ]
        }
    ]
}
```

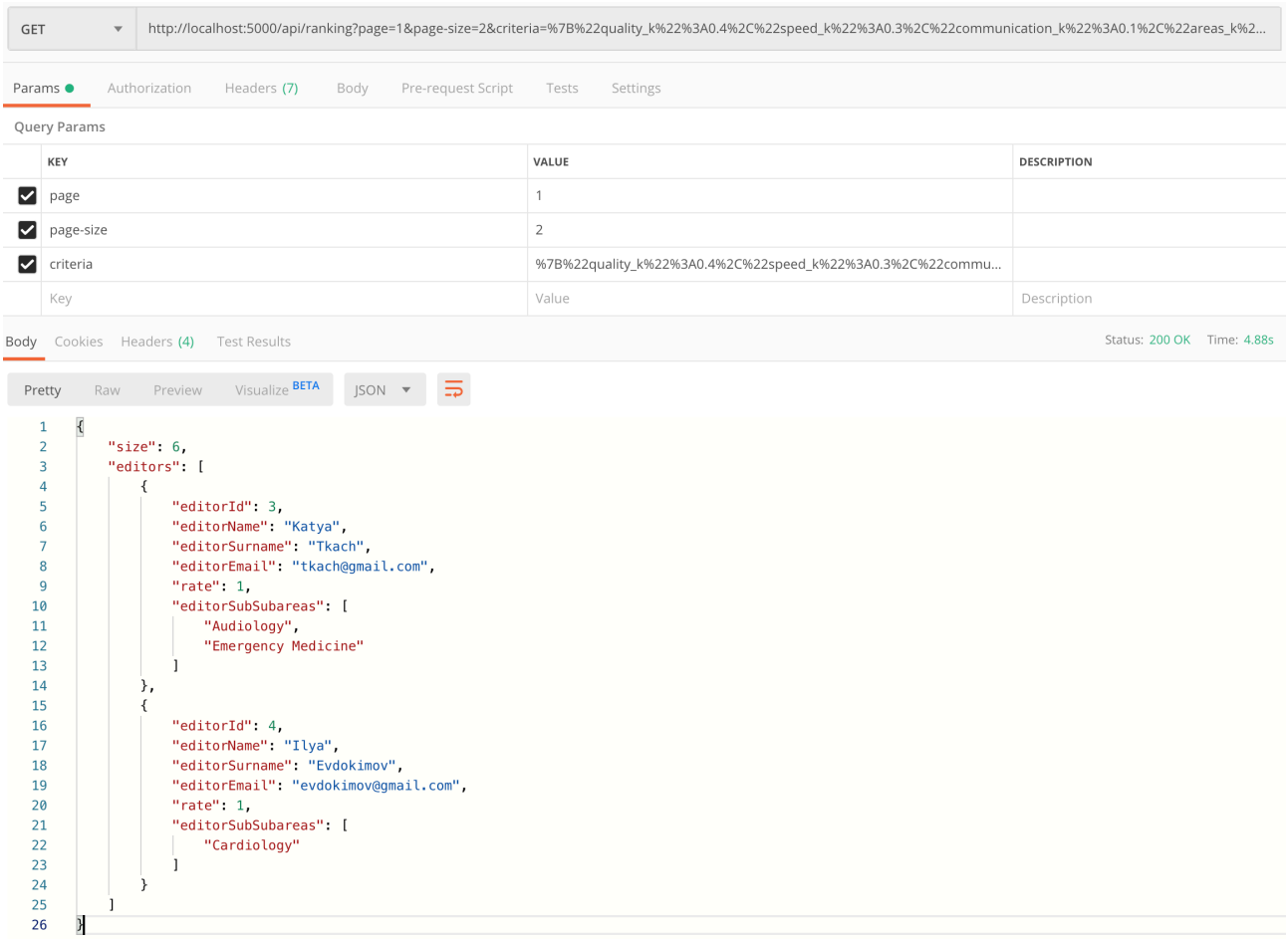



Рис. 3.5 Запит ранжування з критеріями

2) Другий запит на ранжування з пошуком:

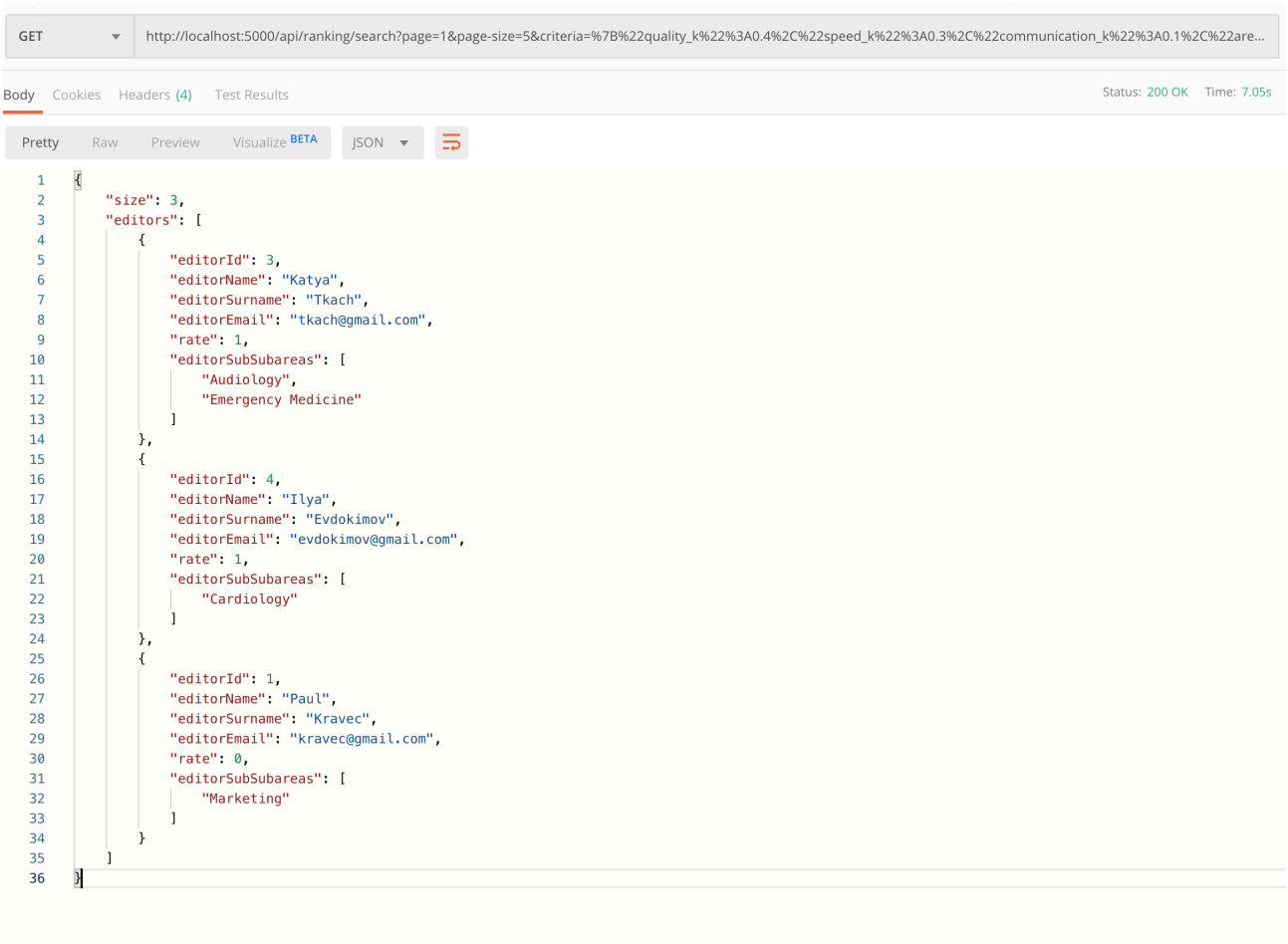
http://localhost:5000/api/ranking/search?page=1&page-size=5&criteria=%7B%22quality_k%22%3A0.4%2C%22speed_k%22%3A0.3%2C

%22communication_k%22%3A0.1%2C%22areas_k%22%3A%5B%7B%22id%22%3A1%2C%22area_k%22%3A0.3%7D%2C%7B%22id%22%3A2%2C%22area_k%22%3A0.25%7D%2C%7B%22id%22%3A3%2C%22area_k%22%3A0.2%7D%2C%7B%22id%22%3A4%2C%22area_k%22%3A0.15%7D%2C%7B%22id%22%3A5%2C%22area_k%22%3A0.1%7D%5D%2C%22ssubareas_checked%22%3A%5B0%2C4%2C5%2C7%2C8%5D%7D&search=1

Отримаємо результат:


```
{
   "size": 3,
   "editors": [
      {
         "editorId": 3,
         "editorName": "Katya",
         "editorSurname": "Tkach",
         "editorEmail": "tkach@gmail.com",
         "rate": 1,
         "editorSubSubareas": [
            "Audiology",
            "Emergency Medicine"
         ]
      },
      {
         "editorId": 4,
         "editorName": "Ilya",
         "editorSurname": "Evdokimov",
         "editorEmail": "evdokimov@gmail.com",
         "rate": 1,
         "editorSubSubareas": [
            "Cardiology"
         ]
```

```
        },
        {
            "editorId": 1,
            "editorName": "Paul",
            "editorSurname": "Kravec",
            "editorEmail": "kravec@gmail.com",
            "rate": 0,
            "editorSubSubareas": [
                "Marketing"
            ]
        }
    ]
}
```


```
{
    "size": 3,
    "editors": [
        {
            "editorId": 3,
            "editorName": "Katya",
            "editorSurname": "Tkach",
            "editorEmail": "tkach@gmail.com",
            "rate": 1,
            "editorSubSubareas": [
                "Audiology",
                "Emergency Medicine"
            ]
        },
        {
            "editorId": 4,
            "editorName": "Ilya",
            "editorSurname": "Evdokimov",
            "editorEmail": "evdokimov@gmail.com",
            "rate": 1,
            "editorSubSubareas": [
                "Cardiology"
            ]
        },
        {
            "editorId": 1,
            "editorName": "Paul",
            "editorSurname": "Kravec",
            "editorEmail": "kravec@gmail.com",
            "rate": 0,
            "editorSubSubareas": [
                "Marketing"
            ]
        }
    ]
}
```


Рис. 3.6 Запит на ранжування з пошуком

Веб-API було протестоване за наданими функціональними вимогами та поставленим завданням. Воно працює без помилок при різних вказаних даних та вірно виконує необхідні обрахування.

Висновки

У результаті даного дослідження можна навести наступні висновки, беручи до уваги постановку завдання курсової роботи.

Був проведений аналіз предметної області, стану розвитку систем ранжування та критеріїв оцінки роботи редакторів. Незалежно від того що є безліч сервісів оцінки та виправлення тексту, праця редакторів надалі залишається актуальною, а засоби ранжування дозволяють оцінити ефективність їх роботи.

Були розглянуті різні шляхи обрахування рейтингу та методи створення Веб-API, була реалізована веб-система для надання функціоналу ранжування редакторів, що оцінюються авторами.

Були проаналізовані можливі засоби розробки, для створення веб-системи було використано ASP.NET фреймворк. Застосунок розроблено відповідно до технічного завдання та протестовано.

Список використаної літератури

1. Entity Framework [Електронний ресурс]
   Режим доступу до статей : https://metanit.com/sharp/entityframework/1.1.php
2. Database First [Електронний ресурс]
   Режим доступу до статей : https://metanit.com/sharp/entityframework/2.4.php
3. Reconstructing [Електронний ресурс] Режим доступу:
   https://docs.microsoft.com/ru-ru/ef/core/managing-schemas/scaffolding
4. Wilson Lower bound Score and Bayesian Approximation for K star scale rating to Rate products [Електронний ресурс] Режим доступу:
   https://medium.com/tech-that-works/wilson-lower-bound-score-and-bayesian-approximation-for-k-star-scale-rating-to-rate-products-c67ec6e30060
5. Bayesian Average Ratings [Електронний ресурс]
   Режим доступу: https://www.evanmiller.org/bayesian-average-ratings.html
6. Bayesian ranking of items with up and downvotes or 5 star ratings [Електронний ресурс]
   Режим доступу: http://julesjacobs.github.io/2015/08/17/bayesian-scoring-of-ratings.html
7. Баєсова ймовірність [Електронний ресурс] Режим доступу:
   https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D1%94%D1%81%D0%BE%D0%B2%D0%B0_%D0%B9%D0%BC%D0%BE%D0%B2%D1%96%D1%80%D0%BD%D1%96%D1%81%D1%82%D1%8C
8. Как правильно сортировать контент на основе оценок пользователей [Електронний ресурс]
   Режим доступу: https://habr.com/ru/company/darudar/blog/143188/
9. Универсальный рейтинг или нам важен каждый голос [Електронний ресурс]
   Режим доступу: https://habr.com/ru/post/172065/
10. Configuring Entity Framework Core [Електронний ресурс]
    Режим доступу: https://www.learnentityframeworkcore.com

11. Database First [Електронний ресурс]

Режим доступу до статей : https://metanit.com/sharp/entityframework/2.4.php

Додатки

Додаток А. Лістинг програмного коду серверної частини

Код класу-моделі Editors.cs

```
using System;
using System.Collections.Generic;

namespace RankingApi.ModelsDB
{
    public partial class Editors
    {
        public Editors()
        {
            EditorsHasSubSubareas = new HashSet<EditorsHasSubSubareas>();
        }
        public int EditorId { get; set; }
        public string EditorName { get; set; }
        public string EditorSurname { get; set; }
        public string EditorEmail { get; set; }
        public DateTime RegistryDate { get; set; }

        public virtual ICollection<EditorsHasSubSubareas> EditorsHasSubSubareas { get; set; }
    }
}
```

Код класу-моделі Rankings.cs

```
using System;
using System.Collections.Generic;
```

```
namespace RankingApi.ModelsDB
{
    public partial class Rankings
    {
        public Rankings()
        {
            Tasks = new HashSet<Tasks>();
        }
        public int IdRanking { get; set; }
        public int Literacy { get; set; }
        public int IndustryConformity { get; set; }
        public int SemanticIntegrity { get; set; }
        public int TextCoherence { get; set; }
        public int TextStructure { get; set; }
        public int InfoSaturation { get; set; }
        public int Communication { get; set; }
        public int Speed { get; set; }
        public int FkSsubareaIdEs { get; set; }
        public int FkEditorIdEs { get; set; }

        public virtual ICollection<Tasks> Tasks { get; set; }
    }
}
```

Код класу-моделі AreaRankingTaskDto.cs

```
namespace RankingApi.ModelsDB
{
    public partial class AreaRankingTaskDto
```

```
    {
        public int IdRanking { get; set; }
        public int Literacy { get; set; }
        public int IndustryConformity { get; set; }
        public int SemanticIntegrity { get; set; }
        public int TextCoherence { get; set; }
        public int TextStructure { get; set; }
        public int InfoSaturation { get; set; }
        public int Communication { get; set; }
        public int Speed { get; set; }
        public int IdTasks { get; set; }
        public int NPages { get; set; }
        public int NDays { get; set; }
        public int EditorId { get; set; }
        public int IdArea { get; set; }
        public int IdSsubarea { get; set; }
    }
}
```

Код класу RankingContext.cs

```
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;

namespace RankingApi.ModelsDB
{
    public partial class RankingContext : DbContext
    {
```

```
        public RankingContext()
        {
        }

        public RankingContext(DbContextOptions<RankingContext> options)
            : base(options)
        {
        }

        public virtual DbSet<Areas> Areas { get; set; }
        public virtual DbSet<Editors> Editors { get; set; }
        public virtual DbSet<EditorsHasSubSubareas> EditorsHasSubSubareas { get; set; }
        public virtual DbSet<Rankings> Rankings { get; set; }
        public virtual DbSet<SubSubareas> SubSubareas { get; set; }
        public virtual DbSet<Subareas> Subareas { get; set; }
        public virtual DbSet<Tasks> Tasks { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            if (!optionsBuilder.IsConfigured)
            {
optionsBuilder.UseMySql("server=localhost;port=3306;database=rankingdb_schema;user=root;pas
sword=****",
                    x => x.ServerVersion("8.0.19-mysql"));
            }
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
```

```
{
    modelBuilder.Entity<Areas>(entity =>
    {
        entity.HasKey(e => e.IdArea)
            .HasName("PRIMARY");

        entity.ToTable("areas");

        entity.Property(e => e.IdArea).HasColumnName("id_area");

        entity.Property(e => e.AreaTitle)
            .IsRequired()
            .HasColumnName("area_title")
            .HasColumnType("varchar(100)")
            .HasCharSet("utf8mb4")
            .HasCollation("utf8mb4_0900_ai_ci");
    });

    modelBuilder.Entity<Editors>(entity =>
    {
        entity.HasKey(e => e.EditorId)
            .HasName("PRIMARY");

        entity.ToTable("editors");

        entity.Property(e => e.EditorEmail)
            .IsRequired()
            .HasColumnType("varchar(100)")
```

```
        .HasCharSet("utf8mb4")
        .HasCollation("utf8mb4_0900_ai_ci");

    entity.Property(e => e.EditorName)
        .IsRequired()
        .HasColumnType("varchar(100)")
        .HasCharSet("utf8mb4")
        .HasCollation("utf8mb4_0900_ai_ci");

    entity.Property(e => e.EditorSurname)
        .IsRequired()
        .HasColumnType("varchar(100)")
        .HasCharSet("utf8mb4")
        .HasCollation("utf8mb4_0900_ai_ci");

    entity.Property(e => e.RegistryDate).HasColumnType("date");
});

modelBuilder.Entity<EditorsHasSubSubareas>(entity =>
{
    entity.HasKey(e => new { e.FkSsubareaId, e.FkEditorId })
        .HasName("PRIMARY");

    entity.ToTable("editors_has_sub_subareas");

    entity.HasIndex(e => e.FkEditorId)
        .HasName("fk_editors_has_sub_subareas_editors1_idx");
```

```
    entity.Property(e => e.FkSsubareaId).HasColumnName("fk_ssubarea_id");

    entity.Property(e => e.FkEditorId).HasColumnName("fk_editor_id");

    entity.HasOne(d => d.Editor)
        .WithMany(p => p.EditorsHasSubSubareas)
        .HasForeignKey(d => d.FkEditorId)
        .HasConstraintName("fk_editors_has_sub_subareas_editors1");

    entity.HasOne(d => d.Ssubarea)
        .WithMany(p => p.EditorsHasSubSubareas)
        .HasForeignKey(d => d.FkSsubareaId)
        .HasConstraintName("fk_editors_has_sub_subareas_sub_subareas1");
});

modelBuilder.Entity<Rankings>(entity =>
{
    entity.HasKey(e => e.IdRanking)
        .HasName("PRIMARY");

    entity.ToTable("rankings");

    entity.HasIndex(e => new { e.FkEditorIdEs, e.FkSsubareaIdEs })
        .HasName("fk_rankings_editors_has_sub_subareas1_idx");

    entity.Property(e => e.IdRanking).HasColumnName("id_ranking");

    entity.Property(e => e.FkEditorIdEs).HasColumnName("fk_editor_id_es");
```

```
    entity.Property(e => e.FkSsubareaIdEs).HasColumnName("fk_ssubarea_id_es");
});

modelBuilder.Entity<SubSubareas>(entity =>
{
    entity.HasKey(e => e.IdSsubarea)
        .HasName("PRIMARY");

    entity.ToTable("sub_subareas");

    entity.HasIndex(e => e.FkSubareaId)
        .HasName("fk_sub_subareas_subareas1_idx");

    entity.Property(e => e.IdSsubarea).HasColumnName("id_ssubarea");

    entity.Property(e => e.FkSubareaId).HasColumnName("fk_subarea_id");

    entity.Property(e => e.SsubareaTitle)
        .IsRequired()
        .HasColumnName("ssubarea_title")
        .HasColumnType("varchar(45)")
        .HasCharSet("utf8mb4")
        .HasCollation("utf8mb4_0900_ai_ci");

    entity.HasOne(d => d.Subarea)
        .WithMany(p => p.SubSubareas)
        .HasForeignKey(d => d.FkSubareaId)
```

```
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("fk_sub_subareas_subareas1");
});

modelBuilder.Entity<Subareas>(entity =>
{
    entity.HasKey(e => e.IdSubarea)
        .HasName("PRIMARY");

    entity.ToTable("subareas");

    entity.HasIndex(e => e.FkAreaId)
        .HasName("fk_subareas_areas1_idx");

    entity.Property(e => e.IdSubarea).HasColumnName("id_subarea");

    entity.Property(e => e.FkAreaId).HasColumnName("fk_area_id");

    entity.Property(e => e.SubareaTitle)
        .IsRequired()
        .HasColumnName("subarea_title")
        .HasColumnType("varchar(100)")
        .HasCharSet("utf8mb4")
        .HasCollation("utf8mb4_0900_ai_ci");

    entity.HasOne(d => d.Area)
        .WithMany(p => p.Subareas)
        .HasForeignKey(d => d.FkAreaId)
```

```
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("fk_subareas_areas1");
});

modelBuilder.Entity<Tasks>(entity =>
{
    entity.HasKey(e => e.IdTasks)
        .HasName("PRIMARY");

    entity.ToTable("tasks");

    entity.HasIndex(e => e.FkRanking)
        .HasName("fk_tasks_rankings1_idx");

    entity.Property(e => e.IdTasks).HasColumnName("id_tasks");

    entity.Property(e => e.FkRanking).HasColumnName("fk_ranking");

    entity.Property(e => e.NDays).HasColumnName("n_days");

    entity.Property(e => e.NPages).HasColumnName("n_pages");

    entity.HasOne(d => d.FkRankingNavigation)
        .WithMany(p => p.Tasks)
        .HasForeignKey(d => d.FkRanking)
        .OnDelete(DeleteBehavior.ClientSetNull)
        .HasConstraintName("fk_tasks_rankings");
});
```

```
            OnModelCreatingPartial(modelBuilder);
        }
        partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
    }
}
```

Код класу RankingController.cs

```
using System;
using System.Web;
using System.Collections.Generic;
using Newtonsoft.Json;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RankingApi.ModelsDB;

namespace RankingApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class RankingController : ControllerBase
    {
        private readonly RankingContext _context;

        public RankingController(RankingContext context)
        {
            _context = context;
        }
```

```
[HttpGet]
public Responce GetEditorsRating(
    [FromQuery(Name = "page")] string page,
    [FromQuery(Name = "page-size")] string pageSize,
    [FromQuery(Name = "criteria")] string criteria
){
    var decoded = HttpUtility.UrlDecode(criteria);
    Criteria criteriaObject = JsonConvert.DeserializeObject<Criteria>(decoded);
    int pageInt = Int32.Parse(page)-1;
    int pageSizeInt = Int32.Parse(pageSize);
    int skip = pageInt*pageSizeInt;
    double[] coefs = {criteriaObject.quality_k, criteriaObject.speed_k, criteriaObject.communication_k};
    var allinfo = ( from r in _context.Rankings
                join t in _context.Tasks on r.IdRanking equals t.FkRanking
                join se in _context.EditorsHasSubSubareas on r.FkEditorIdEs equals se.FkEditorId
                join e in _context.Editors on se.FkEditorId equals  e.EditorId
                join ss in _context.SubSubareas on se.FkSsubareaId equals ss.IdSsubarea
                join s in _context.Subareas on ss.FkSubareaId equals s.IdSubarea
                where r.FkSsubareaIdEs == se.FkSsubareaId &&
criteriaObject.ssubareas_checked.Contains(ss.IdSsubarea)
                select new AreaRankingTaskDto{
                    IdRanking = r.IdRanking,
                    Literacy = r.Literacy,
                    IndustryConformity = r.IndustryConformity,
                    SemanticIntegrity  = r.SemanticIntegrity,
                    TextCoherence = r.TextCoherence,
                    TextStructure = r.TextStructure,
                    InfoSaturation = r.InfoSaturation,
                    Communication = r.Communication,
                    Speed = r.Speed,
```

```
                    IdTasks = t.IdTasks,
                    NPages = t.NPages,
                    NDays = t.NDays,
                    EditorId = e.EditorId,
                    IdArea = s.FkAreaId,
                    IdSsubarea = ss.IdSsubarea,
                }
    ).ToList();

    var result = _context.Editors.Select( editor => new RatingDto{
        EditorId = editor.EditorId,
        EditorName = editor.EditorName,
        EditorSurname = editor.EditorSurname,
        EditorEmail = editor.EditorEmail,
        EditorSubSubareas = editor.EditorsHasSubSubareas
                    .Where(es => es.FkEditorId == editor.EditorId)
                    .Select(s => s.Ssubarea.SsubareaTitle).ToList()
    }).ToList();

    foreach (RatingDto ratingDto in result) {
        Console.WriteLine("Id: ", ratingDto.EditorId);
        var arr = allinfo.Where(ai => ai.EditorId == ratingDto.EditorId).ToArray();
        Console.WriteLine("N: ", arr.Length);
        ratingDto.Rate = FinalRate(arr, criteriaObject.areas_k, coefs);
        Console.WriteLine("Rate: ",ratingDto.Rate);
    }
    var count = result.Count();
    var resultarr = result.OrderByDescending(r => r.Rate).Skip(skip).Take(pageSizeInt).ToList();

    return new Responce(count, resultarr);
}
```

```
        [HttpGet]
        [Route("search")]
        public Responce GetEditorsRatingSearch(
            [FromQuery(Name = "page")] string page,
            [FromQuery(Name = "page-size")] string pageSize,
            [FromQuery(Name = "criteria")] string criteria,
            [FromQuery(Name = "search")] string search
        ){
            var decoded = HttpUtility.UrlDecode(criteria);
            Criteria criteriaObject = JsonConvert.DeserializeObject<Criteria>(decoded);
            int pageInt = Int32.Parse(page)-1;
            int pageSizeInt = Int32.Parse(pageSize);
            int skip = pageInt*pageSizeInt;
            double[] coefs = {criteriaObject.quality_k, criteriaObject.speed_k, criteriaObject.communication_k};
            var allinfo = ( from r in _context.Rankings
                        join t in _context.Tasks on r.IdRanking equals t.FkRanking
                        join se in _context.EditorsHasSubSubareas on r.FkEditorIdEs equals se.FkEditorId
                        join e in _context.Editors on se.FkEditorId equals  e.EditorId
                        join ss in _context.SubSubareas on se.FkSsubareaId equals ss.IdSsubarea
                        join s in _context.Subareas on ss.FkSubareaId equals s.IdSubarea
                        where r.FkSsubareaIdEs == se.FkSsubareaId &&
criteriaObject.ssubareas_checked.Contains(ss.IdSsubarea)
                        select new AreaRankingTaskDto{
                            IdRanking = r.IdRanking,
                            Literacy = r.Literacy,
                            IndustryConformity = r.IndustryConformity,
                            SemanticIntegrity  = r.SemanticIntegrity,
                            TextCoherence = r.TextCoherence,
                            TextStructure = r.TextStructure,
                            InfoSaturation = r.InfoSaturation,
                            Communication = r.Communication,
                            Speed = r.Speed,
```

```
                IdTasks = t.IdTasks,
                NPages = t.NPages,
                NDays = t.NDays,
                EditorId = e.EditorId,
                IdArea = s.FkAreaId,
                IdSsubarea = ss.IdSsubarea,
            }
).ToList();

var result = _context.Editors.Select( editor => new RatingDto{
    EditorId = editor.EditorId,
    EditorName = editor.EditorName,
    EditorSurname = editor.EditorSurname,
    EditorEmail = editor.EditorEmail,
    EditorSubSubareas = editor.EditorsHasSubSubareas
                .Where(es => es.FkEditorId == editor.EditorId)
                .Select(s => s.Ssubarea.SsubareaTitle).ToList()
}).ToList();

foreach (RatingDto ratingDto in result) {
    Console.WriteLine("Id: ", ratingDto.EditorId);
    var arr = allinfo.Where(ai => ai.EditorId == ratingDto.EditorId).ToArray();
    Console.WriteLine("N: ", arr.Length);
    ratingDto.Rate = FinalRate(arr, criteriaObject.areas_k, coefs);
    Console.WriteLine("Rate: ",ratingDto.Rate);
}
var searchresult = result.Where(t =>
                t.EditorId.ToString().Contains(search) ||
                t.EditorName.Contains(search) ||
                t.EditorSurname.Contains(search) ||
                t.EditorEmail.Contains(search) ||
                t.EditorSubSubareas.Contains(search)||
```

```
                    t.Rate.ToString().Contains(search))
                .ToList();
        var count = searchresult.Count;
        var resultarr = searchresult.OrderByDescending(r => r.Rate).Skip(skip).Take(pageSizeInt).ToList();
        return new Responce(count, resultarr);
    }

    private double FinalRate(AreaRankingTaskDto[] rankings, List<AreaK> area_coefs, double[] coefs){
        List<double> ranks = new List<double>();
        foreach (AreaRankingTaskDto r in rankings){
            double[] marks = {r.IndustryConformity,r.InfoSaturation,r.Literacy,r.SemanticIntegrity, r.TextCoherence,
r.TextStructure};
            double quality = GeomAverage(marks);
            double communication = r.Communication;
            double area_coef = area_coefs.Find(x => x.id == r.IdArea).area_k;
            double speed = Speed(r.NPages, r.NDays, r.Speed, area_coef);
            double weightedSum = WeightedSum(quality, communication, speed, coefs);
            ranks.Add(weightedSum);
        }
        foreach(double rank in ranks){
            Console.WriteLine(rank);
        }
        return Rate(ranks.ToArray());
    }

    private double MeanRate(double[] weightedMarks) {
        double sum = 0;
        for (int i = 0; i < weightedMarks.Length; i++)
            sum += weightedMarks[i];

        return sum / weightedMarks.Length;
    }
```

```
private double GeomAverage(double[] arr){
    double product = 1;
    int n = arr.Length;
    for (int i = 0; i < n; i++)
        product = product * arr[i];

    double gm = (double)Math.Pow(product, (double)1 / n);
    return gm;
}
private double Speed(int nPages, int nDays, double speed_mark, double coef){
    double actual_speed = (nPages/nDays)>=15 ? 1 : 0;
    return speed_mark * (actual_speed+coef);
}
private double WeightedSum(double quality,double communication,double speed, double[] coefs){
    return quality*coefs[0]+speed*coefs[1]+communication*coefs[2];
}
private double Rate(double[] rates){
    double N = rates.Sum();
    int K = rates.Length;
    double z = 1.959963;
    if (N == 0) return 0;
    double first_part = 0.0;
    double second_part = 0.0;
    for(int k=0; k < K; k++){
        first_part += (k+1)*(rates[k]+1)/(N+K);
        second_part += (k+1)*(k+1)*(rates[k]+1)/(N+K);
    }
    double score = first_part - z * Math.Sqrt((second_part - first_part*first_part)/(N+K+1));
    return score;
}
```

```
    }
}
```